



INSTITUTE FOR DEFENSE ANALYSES

## **The Software Assurance State-of-the-Art Resource (SOAR)**

E. Kenneth Hong Fong, *Project Leader*

David A. Wheeler

August 2017

Approved for public  
release; distribution is  
unlimited.

IDA Non-Standard  
NS D-8462

Log: H 2017-000254

INSTITUTE FOR DEFENSE  
ANALYSES  
4850 Mark Center Drive  
Alexandria, Virginia 22311-1882



*The Institute for Defense Analyses is a non-profit corporation that operates three federally funded research and development centers to provide objective analyses of national security issues, particularly those requiring scientific and technical expertise, and conduct related research on other national challenges.*

#### About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-14-D-0001, Task AU-5-3856, "Software Assurance State-of-the-Art Resources," for Office of the Deputy Assistant Secretary of Defense for Systems Engineering, Acquisition Technology and Logistics. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

#### For more information:

David A. Wheeler, Project Leader  
dwheeler@ida.org, 703-845-6662

Margaret E. Myers, Director, Information Technology and Systems Division  
mmyers@ida.org, 703-578-2782

#### Copyright Notice

© 2017 Institute for Defense Analyses  
4850 Mark Center Drive, Alexandria, Virginia 22311-1882 • (703) 845-2000.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (a)(16) [Jun 2013].

# The Software Assurance State-of-the-Art Resource

David A. Wheeler

2017-08-25

INSTITUTE FOR DEFENSE ANALYSES

*Unintentional and intentionally inserted vulnerabilities in software can provide adversaries with various avenues to reduce system effectiveness, render systems useless, or even use our systems against us. Unfortunately, it can be difficult to determine what types of tools and techniques exist for evaluating software, and where their use is appropriate. The State-of-the-Art Resource for Software Vulnerability Detection, Test, and Evaluation, a.k.a. the “Software SOAR,” was written to enable program managers and their staffs to make effective software assurance and software supply chain risk management (SCRM) decisions, particularly when they are developing and executing their program protection plans (PPP). A secondary purpose is to inform DoD policymakers who are developing software policies. This paper summarizes the Software SOAR, including some of the over 50 types of tools and techniques available, and an overall process for selecting and using appropriate analysis tool/technique types for evaluating software. It also discusses some of the changes made in its latest update.*

## 1. Introduction

Nearly all modern systems depend on software. It may be embedded within the system, delivering capability; used in the design and development of the system; or used to manage and control the system, possibly through other systems. Software may be acquired as a commercial-off-the-shelf (COTS) component or may be custom-developed for the system. Software is often embedded within subcomponents by manufacturers. Modern systems often perform the majority of their functions through software, which can easily include millions of lines of software code.

Although functionality is often created through software, this software can also introduce risks. Unintentional or intentionally inserted vulnerabilities (including previously known vulnerabilities) can provide adversaries with various avenues to reduce system effectiveness, render systems useless, or even turn our systems against us. Department of Defense (DoD) software, in particular, is subject to attack.

This is emphasized in the February 2017 edition of DoD Instruction 5000.02, Enclosure 14, *Cybersecurity in the Defense Acquisition System*. This enclosure states that, “Cybersecurity is a requirement for all DoD programs and must be fully considered and implemented in all aspects of acquisition programs across the life cycle... Program managers...are responsible for the cybersecurity of their programs, systems, and information [from] the earliest exploratory phases...through all phases of the acquisition. ... Program Managers will...request assistance, when appropriate, from the Joint

Federated Assurance Center...to support software and hardware assurance requirements...[and] Incorporate automated software vulnerability analysis tools throughout the life cycle to evaluate software vulnerabilities....” [DoDI 5000.02 2017]

In short, analyzing DoD software to identify and remove weaknesses is a critical program protection countermeasure. What is more, because of its scale, it is often impractical to analyze software using purely manual approaches.

Unfortunately, it can be difficult to determine what types of tools and manual techniques exist for analyzing software, and where their use is appropriate. Even many software developers are unaware of the many types of tools and techniques available. Tool developers may emphasize how their tool is different from all other tools, resulting in more confusion by those trying to understand the different types of tools available.

To help reduce the confusion, our IDA team developed a document we call the “Software SOAR” or “Software Assurance SOAR” (its full title is *State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation*). The Software SOAR was originally released to the public in 2014 [Wheeler2014], and an updated version is now available [Wheeler2016].

The purpose of the Software SOAR is to assist DoD program managers (PM), and their staffs, in making effective software assurance (SwA) and software supply chain risk management (SCRM) decisions, particularly when they are developing their program protection plans (PPP). A secondary purpose is to inform DoD policymakers who are developing software policies.

In this paper, we first highlight the overall process we recommend for selecting and reporting results from appropriate tools and techniques. This process depends on projects identifying their technical objectives, which we discuss. The next section discusses the various types of tools and techniques available to help meet those objectives, followed by a section about changes in the SOAR.

The Software SOAR includes other information not discussed here, including gaps that were identified, key topics raised in interviews, detailed fact sheets, and the impact of the mobile environment. See the Software SOAR for more information.

## **2. Overall Process for Selecting and Reporting Results from Appropriate Tools and Techniques**

Our proposed approach for selecting various tools and techniques, and developing reports using them, is to first identify the software components in a target of evaluation (TOE) and determine each software component’s context of use. Then, for each software component context of use:

1. Identify technical objectives based on context.

2. Select tool/technique types needed to address the technical objectives, using the matrix discussed below.
3. Select specific tools and techniques of the relevant types.
4. Summarize selection (write down your plan), which may be part of a larger report. Within DoD, this would be part of the PPP.
5. Apply the analysis tools, use their results, and report appropriately. Here, the selected tools and techniques are applied, including the selection, modification, or risk mitigation of software based on tool/technique results, and reports are provided to those with oversight authority.

Since different tool/technique types are better at addressing different technical objectives, we suggest ensuring that the set of tools/techniques selected adequately cover the intended technical objectives. One way to do this is to use a matrix we have developed that specifies the technical objectives met, to some degree, by various tool/technique types. The table below illustrates this matrix. The table cells indicate the applicability, e.g., a checkmark with a yellow cell indicates that the tool/technique type can be a highly cost-effective measure to address this technical objective and should be investigated further. A green circled checkmark indicates the tool/technique type completely addresses this technical objective (unfortunately, this is rare).

Technical Objective	Lower-level technical objective	Need indicator	Tool/technique type									
			Static			Dynamic			Hybrid			
			1	2	...	21	22	...	31	...		
Design & code quality												
Counter unintentional-like known vulnerabilities	...			√								
Authentication & access control	Authentication											
	...						√					
Counter unintentional-like weaknesses	Buffer handling	C/C++/ Objective-C										
...	...											

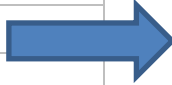
### 3. Technical Objectives

Different types of tools and techniques are better for different purposes. Thus, it is important to identify the various purposes for using different types of tools and techniques, so that the most appropriate types can be selected. The Software SOAR terms these purposes “technical objectives.”

It is common for security issues to be categorized as being related to confidentiality, integrity, and availability; DoD also separately considers authentication and non-repudiation [DoDI 8500.01]. However, since a vulnerability can cause problems in all of those areas, these categorizations are too general to support narrowing the selection of appropriate tool/technique types.

Even at a more detailed level, there is no universally accepted set of categories for technical objectives. The Common Weakness Enumeration (CWE) identifies a very large set of common weaknesses in software that may lead to vulnerabilities, but while CWE is useful for many purposes, it does not provide a single, simple organizational structure. “Top” lists, such as the “CWE/SANS top 25” and the “Open Web Application Security Project (OWASP) top 10,” are helpful in identifying especially common weaknesses, but they make no attempt to cover all relevant objectives.

Instead, we have focused on identifying a set of detailed technical objectives that can help narrow the selection of appropriate tools and techniques. We created this set of technical objectives by merging several accepted sources. Here is the top-level set of technical objectives:

1. Provide design and code* quality		1. Buffer Handling*
2. Counter unintentional-like known vulnerabilities		2. Injection* (SQL, command, etc.)
3. Ensure authentication and access control* <ul style="list-style-type: none"> <li>a. Authentication Issues</li> <li>b. Credentials Management</li> <li>c. Permissions, Privileges, and Access Control</li> <li>d. Least Privilege</li> </ul>		3. Encryption and Randomness*
4. Counter unintentional-“like” weaknesses		4. File Handling*
5. Counter intentional-“like”/malicious logic* <ul style="list-style-type: none"> <li>a. Known malware</li> <li>b. Not known malware</li> </ul>		5. Information Leaks*
6. Provide anti-tamper and ensure transparency		6. Number Handling*
7. Counter development tool inserted weaknesses		7. Control flow management*
8. Provide secure delivery		8. Initialization and Shutdown [of resources/ components]*
9. Provide secure configuration		9. Design Error
10. Other		10. System Element Isolation
		11. Error Handling* a Fault isolation
		12. Pointer and reference handling*

#### **4. Types of Tools and Techniques**

There is no widely accepted complete categorization of tools and techniques. For example, the National Institute of Standards and Technology Software Assurance Metrics and Tool Evaluation (NIST SAMATE) project web page has a brief but limited list of tool categories. This lack of categorization is one reason why the space is so confusing.

We have created a categorization of tools and techniques based on our own analysis, using sources such as interviews and the NIST SAMATE project. It is not the only possible categorization, and since it is incomplete, we do not call it a taxonomy. Our goal is simply to create a useful set of categories that can be extended as required. In general, we only included tool types where there is at least one commercially available tool; we granted some exceptions in the mobile space because that is a fast-paced environment. We expect that new types of tools and technologies could be added in the future to these categories, driven by innovation and commercialization (especially in the mobile environment).

The latest version of the SOAR identifies 59 types of tools and techniques available for analyzing software. We have identified the following three major groups of tool/technique types:

- **Static analysis:** Examines the system/software without executing it, including examining source code, bytecode, and/or binaries.
- **Dynamic analysis:** Examines the system/software by executing it, giving it specific inputs, and examining results and/or outputs.
- **Hybrid analysis:** Tightly integrates static and dynamic analysis approaches; for example, test coverage analyzers use dynamic analysis to run tests and then use static analysis to determine which parts of the software were not tested. This grouping is used only if static and dynamic analyses are tightly integrated; a tool or technology type that is primarily static or primarily dynamic is put in those groupings instead.

The following sections identify a subset of tool/technology types in each of these three major groups.

##### **4.1. Static Analysis**

Here are some of the common static analysis tool/technology types:

1. **Attack modeling.** Attack modeling analyzes the system architecture from an attacker's point of view to find weaknesses or vulnerabilities that should be countered.

2. Source code analyzers<sup>1</sup> is a group of the following tool types:
  - a. Warning flags. Warning flags are mechanisms built into programming language implementations and platforms that warn of dangerous circumstances while processing source code.
  - b. Source code quality analyzer. Source code quality analyzers examine software source code and search for the implementation of poor coding or certain poor architecture practices, using pattern matches against good coding practices or mistakes that can lead to poor functionality, poor performance, costly maintenance, or security weaknesses, depending on context. There is now a preponderance of evidence that higher-quality software (in general) tends to produce more secure software [Woody 2014]. These kinds of tools are often less expensive than some other kinds, and can often be applied earlier in development, providing good reasons to use them even when the focus is to develop secure software.
  - c. Source code weakness analyzer. Source code weakness analyzers examine software source code and search for vulnerabilities, using pattern matches against well-known common types of vulnerabilities (weaknesses). This kind of tool is also called a “source code security analyzer,” “static application security testing” (SAST) tool, “static analysis code scanner,” or “code weakness analysis tool.” We’ve chosen the name “source code weakness analyzer” because this name more clearly defines what this type of tool does and distinguishes it from other types of analysis.
  - d. Context-configured source code weakness analyzer. This configures a source code weakness analyzer specifically for the product being evaluated (e.g., by adding many additional rules).
3. Binary/bytecode analysis is a group of the following tool types:
  - a. Traditional virus/spyware scanner. Traditional virus/spyware scanners search for known malicious patterns in the binary or bytecode.
  - b. Quality analyzer. Binary/bytecode quality analyzers examine the binary or bytecode (respectively) and search for the implementation of poor coding or certain poor architecture practices, using pattern matches against good coding practices or mistakes that can lead to poor functionality, performance, costly maintenance, or security weaknesses depending on context.

---

<sup>1</sup> For the purposes of this paper, “source code analyzer” is a *group* of tool types; the lettered items below are the tool/technique types. A person who performs manual review of source code could also be considered a “source code analyzer,” but for our purposes we group manual review processes separately.



- c. Bytecode weakness analyzer. Bytecode weakness analyzers examine binaries and search for vulnerabilities, using pattern matches against well-known common types of vulnerabilities (weaknesses). Note that these are similar to source code weakness analyzers, except that the analysis is performed on bytecode.
  - d. Binary weakness analyzer. Binary weakness analyzers examine binaries and search for vulnerabilities, using pattern matches against well-known common types of vulnerabilities (weaknesses). Note that these are similar to source code weakness analyzers, except that the analysis is performed on a binary.
4. Human review. This is typically done with source code, but it can also be done with binary or bytecode (often this is generated by a binary or bytecode disassembler, as noted above). Note that human reviews can apply to products other than code, including requirements, architecture, design, and test artifacts. Human reviews include the following more-specific types of techniques:
- a. Focused manual spot check. This specialized technique focuses on manual analysis of code (typically less than 100 lines of code) to answer specific questions. For example, does the software require authorization when it should? Do the software interfaces contain input checking and validation?
  - b. Manual code review (other than inspections). This specialized technique is the manual examination of code, e.g., to look for malicious code.
  - c. Inspections (Institute of Electrical and Electronics Engineers (IEEE) standard). IEEE 1028 inspection is a systematic peer examination to detect and identify software product anomalies.
  - d. Generated code inspection. This technique examines generated binary or bytecode to determine that it accurately represents the source code. For example, if a compiler or later process inserts malicious code, this technique might detect it. This is usually a spot check and not performed across all of the code.
5. Secure platform selection is a group of the following tool types:
- a. Safer languages. This is selecting languages, or language subsets, that eliminate or make it more difficult to inadvertently insert vulnerabilities. This includes selecting memory-safe and type-safe languages.
  - b. Secure library selection. Secure libraries provide mechanisms designed to simplify developing secure applications. They may be standalone or be built into larger libraries and platforms.

- c. Secured operating system (OS). A secured OS is an underlying operating system and platform that is hardened to reduce the number, exploitability, and impact of vulnerabilities.
6. Origin analyzer. Origin analyzers are tools that analyze source code, bytecode, or binary code to determine their origins (e.g., pedigree and version). From this information, some estimate of riskiness may be determined, including the potential identification of obsolete/vulnerable libraries and reused code.
7. Digital signature verification. Digital signature verification ensures that software is verified as being from the authorized source (and has not been tampered with since its development). This typically involves checking cryptographic signatures.
8. Configuration checker. Configuration checkers assess the configuration of software to ensure that it meets requirements, including security requirements. A configuration is the set of settings that determine how the software is accessed, is protected, and operates.

#### **4.2. Dynamic Analysis**

Here are some of the common dynamic analysis tool/technology types:

1. Application-type-specific vulnerability scanner. An application-type-specific vulnerability scanner sends data to an application, to identify both known and new vulnerabilities. It may look for known vulnerability patterns (a.k.a. weaknesses) and anomalies. This is a group of the following tool types:
  - a. Web application vulnerability scanner. A web application vulnerability scanner automatically scans web applications for potential vulnerabilities. They typically simulate a web browser user, by trawling through URLs and trying to attack the web application. For example, they may perform checks for field manipulation and cookie poisoning [SAMATE].
  - b. Web services scanner. A web services scanner automatically scans a web service (as opposed to a web application), e.g., for potential vulnerabilities. [SAMATE]
  - c. Database scanner. Database scanners are specialized tools used specifically to identify vulnerabilities in database applications. [SAMATE] For example, they may detect unauthorized altered data (including modification of tables) and excessive privileges.
2. Fuzz tester. A fuzz tester provides invalid, unexpected, or random data to software, to determine whether problems occur (e.g., crashes or failed built-in assertions). Note that many scanners (listed above) use fuzz testing approaches.

3. Automated detonation chamber (limited time) automatically isolates a program (including running multiple copies in virtual machines), executes it, detects potentially malicious or unintentionally vulnerable activities, and then reports its findings prior to the software's deployment. In contrast, we use the broader term "monitored execution" to refer to broader processes that use many tools/techniques (including manual techniques) to isolate software and detect malicious activities.

#### **4.3. Hybrid Analysis**

Here are some of the common hybrid analysis tool/technology types:

4. Test coverage analyzer. Test coverage analyzers are tools that measure the degree to which a program has been tested (e.g., by a regression test suite). Common measures of test coverage include statement coverage (the percentage of program statements executed by at least one test) and branch coverage (the percentage of program branch alternatives executed by at least one test). Areas that have not been tested can then be examined, e.g., to determine whether more tests should be created or whether that code is unwanted.
5. Hardening tools/scripts. This type of tool modifies software configuration to counter or mitigate attacks, or to comply with policy. In the process, it may detect weaknesses or vulnerabilities in the software being configured.
6. Execute and compare with application manifest. Run an application with a variety of inputs to determine the permissions it tries to use, and compare that with the application permission manifest.
7. Track sensitive data. Statically identify data that should not be transmitted or shared (e.g., due to privacy concerns or confidentiality requirements), then dynamically execute the application, tracking that data as tainted to detect exfiltration attempts.
8. Coverage-guided fuzz tester. Use code coverage information to determine new inputs to test.

#### **5. Changes to the SOAR**

In the May 2014 version of the SOAR, we noted that there was a lack of specific quantitative data to support the hypothesis that higher software quality tends to produce more secure software. At the time this was a plausible hypothesis that a number of experts believed to be true. However, many seemingly reasonable hypotheses are false. We believed in 2014 that it was important to investigate this claim before recommending it. This question is important, because if it is true, then it might be appropriate to first use tools to identify quality problems, fix the problems they identify, and then use other tools

for more complex analysis. More evidence that supports this hypothesis has since been published. In particular, SEI [Woody 2014] published in December 2014 a compendium of evidence to support the claim that higher quality software (in a general sense) tends to produce more secure software.

While more evidence would be welcome, we believe the preponderance of evidence now is that improving the general quality of software tends to improve the security of the software. This does not mean that using only generic quality tools is enough to develop secure software. Instead, it means that using generic quality tools can be a valuable aid in developing secure software.

We searched for new types of tools and techniques in the commercial software market (both open source software and proprietary software). Software assurance is not a solved problem, and while most tool suppliers had refined their tools further, we were disappointed that we did not find more new approaches. That said, we added some new tool categories not in previous versions of the SOAR. For example, we added “coverage-guided fuzz tester” as a category to cover tools such as American Fuzzy Lop (an open source software tool that has found a large number of vulnerabilities).

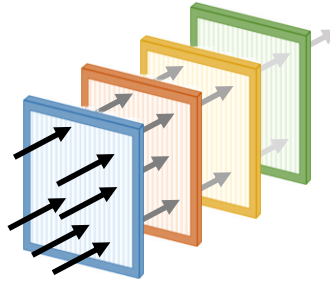
All of these additional types of tools are hybrid approaches, which is interesting because we had previously predicted that more tool types would be created as hybrids to take advantage of the information available from both static and dynamic analysis.

We also added more guidance on how to apply this, including tips on selecting technical objectives and how to select tools and techniques given those technical objectives. We added a mapping to the OWASP top 10 of 2013 (which is widely used when developing web applications). We also discussed other kinds of tools that are related but not the primary focus, such as SwA correlation tools.

## **6. Combining Approaches**

No one type of tool or technique can address all possible technical objectives. Some tool/technique types address only one or a few specific technical objectives, but are highly effective for that scope. Those that have broader applicability may have challenges (e.g., some can be more costly or require deeper expertise).

Thankfully, static, dynamic, and hybrid analysis tools and techniques can be combined to alleviate some of these limitations. The following figure is a conceptual illustration of the advantages of using multiple tools and techniques, particularly when they use different approaches. The arrows represent potential risks, including exposed vulnerabilities in the software, and the screens represent tools and techniques applied by a project. No one tool or technique addresses all technical objectives, and almost all find only a fraction of the vulnerabilities and other issues they address. Each tool or technique contributes to meeting technical objectives (and thus reducing overall risk).



However, achieving the desired result will not happen by accident. Programs need to proactively determine their technical objectives, determine what kinds of tools and techniques will help them achieve their objectives, and then smartly apply these tools and techniques. We hope that the Software SOAR will help programs identify and achieve their objectives.

### **Bibliography**

[DoDI 5000.02 2017] DoD. February 2, 2017 (Change 2). Operation of the Defense Acquisition System. DoD Instruction 5000.02.

[http://www.dtic.mil/whs/directives/corres/pdf/500002\\_dodi\\_2015.pdf](http://www.dtic.mil/whs/directives/corres/pdf/500002_dodi_2015.pdf)

[DoDI 8500.01] DoD. March 14, 2014. Cybersecurity. DoD Instruction 8500.01.

[http://www.dtic.mil/whs/directives/corres/pdf/850001\\_2014.pdf](http://www.dtic.mil/whs/directives/corres/pdf/850001_2014.pdf)

[SAMATE] “Classes of Tools & Techniques.” Retrieved 2017-04-04.

[https://samate.nist.gov/index.php/Tool\\_Survey.html](https://samate.nist.gov/index.php/Tool_Survey.html)

[Wheeler2014] Wheeler, David A. and Rama S. Moorthy. *State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation*. July 2014. IDA Paper P-5061. <http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf>

[Wheeler2016] Wheeler, David A. and Amy E. Henninger. *State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016*. November 2016. IDA Paper P-8005. [http://www.acq.osd.mil/se/initiatives/init\\_jfac.html](http://www.acq.osd.mil/se/initiatives/init_jfac.html)

[Woody2014] Woody, Carol, Robert Eillison, and William Nichols. “Predicting Software Assurance Using Quality and Reliability Measures.” December 2014. Technical Note CMU/SEI-2014-TN-026. Carnegie Mellon University (GMU) Software Engineering Institute (SEI) CERT Division/SSD.



REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188		
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b>					
1. REPORT DATE (DD-MM-YY) 2017-04-04		2. REPORT TYPE Non-Standard		3. DATES COVERED (From – To)	
4. TITLE AND SUBTITLE The Software Assurance State-of-the-Art Resource (SOAR)  CSIAC article			5a. CONTRACT NUMBER HQ0034-14-D-0001		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBERS		
6. AUTHOR(S) David A. Wheeler			5d. PROJECT NUMBER AU-5-3856		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES Institute for Defense Analyses 4850 Mark Center Drive Alexandria, VA 22311-1882			8. PERFORMING ORGANIZATION REPORT NUMBER NS D-8462 H 2017-000254		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Kristen Baldwin, Acting Deputy Secretary of Defense, Systems Engineering Office of the Deputy Assistant Secretary of Defense for Systems Engineering, Acquisition Technology and Logistics 3030 Defense Pentagon, Room 3C167, Washington, DC 20301-3030			10. SPONSOR'S / MONITOR'S ACRONYM DASD SE		
			11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES Project Leader: E. Kenneth Hong Fong					
14. ABSTRACT Unintentional and intentionally inserted vulnerabilities in software can provide adversaries with various avenues to reduce system effectiveness, render systems useless, or even use our systems against us. Unfortunately, it can be difficult to determine what types of tools and techniques exist for evaluating software, and where their use is appropriate. The State-of-the-Art Resource for Software Vulnerability Detection, Test, and Evaluation, a.k.a. the "Software SOAR," was written to enable program managers and their staffs to make effective software assurance and software supply chain risk management (SCRM) decisions, particularly when they are developing and executing their program protection plans (PPP). A secondary purpose is to inform DoD policymakers who are developing software policies. This paper summarizes the Software SOAR, including some of the over 50 types of tools and techniques available, and an overall process for selecting and using appropriate analysis tool/technique types for evaluating software. It also discusses some of the changes made in its latest update.					
15. SUBJECT TERMS Software; software assurance; SOAR; assurance tools; vulnerabilities; security; software development					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT  Unlimited	18. NUMBER OF PAGES  11	19a. NAME OF RESPONSIBLE PERSON Kristen Baldwin, Acting Deputy Secretary of Defense, Systems Engineering
					a. REPORT Unclassified

